SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

PROGRAMA INSTITUCIONAL DE BOLSAS DE INICIAÇÃO CIENTÍFICA – PIBIC

Reconhecimento de padrões biométricos utilizando máquinas de aprendizado profundo: *Elaboração de modelo de máquina de aprendizado sequencial*

Área do conhecimento: Ciência da Computação
Subárea do conhecimento: Metodologia e Técnicas da Computação
Especialidade do conhecimento: Processamento Gráfico (*Graphics*)

Relatório Final

Período da bolsa: de agosto de 2017 a julho de 2018

Este projeto é desenvolvido com bolsa de iniciação científica
PIBIC/COPES

Orientador: Prof. D.Sc. Leonardo Nogueira Matos

Autor: Gabriel Anísio dos Santos Soares

São Cristóvão – Sergipe

2019

Resumo

A interface cérebro-computador é um dos campos emergentes da interação homem-computador devido ao seu amplo espectro de aplicações, especialmente as que lidam com a cognição humana. Neste trabalho, a eletroencefalografia (EEG) é usada como dado base para classificar o estado dos olhos (abertos ou fechados) aplicando redes *Long Short Term Memory* (LSTM) e variantes. Para fins de *benchmarking*, foi utilizado o conjunto de dados de EEG com registro do estado do olho, disponível no repositório de aprendizado de máquina da Universidade da Califórnia em Irvine (UCI). Os resultados obtidos indicaram que o modelo é aplicável para a classificação dos dados e que seu desempenho é bom comparado aos modelos mais caros computacionalmente. Adicionalmente foi desenvolvido um sistema de captura de dados com baseado na plataforma *YouMake*, *hardware* acessível desenvolvido na instituição para ter acesso facilitado a dados biomédicos considerando a possibilidade de obter os dados tanto com os aparelhos convencionais (mas com custos proibitivamente altos), quanto com dispositivo desenvolvido (com custo mais baixo e limitado).

Palavras-chave: EEG, LSTM, Aprendizado de Máquina

Abstract

The brain-computer interface is one of the emerging fields of human-computer interaction due to its broad spectrum of applications, especially those dealing with human cognition. In this assignment, electroencephalography (EEG) signals are used as a base data to classify the state of the eyes (open or closed) by applying Long Short Term Memory (LSTM) networks and variants. For benchmarking purposes, the eye-state EEG data set available at the University of California Irvine (UCI) Machine Learning repository was used. The results indicated that the model is applicable to the classification of the data and that its performance is reasonable compared to the more expensive models computationally. In addition, a data capture system prototype was developed based on the YouMake platform, accessible hardware developed at the institution to have easy access to biomedical data considering the possibility of obtaining data with both conventional (but with prohibitively high costs) and device developed (with lower and limited cost).

Keywords: EEG, LSTM, Machine Learning

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

BGRU	*Bidirectional Gated Recurrent Unit*
BLSTM	*Bidirectional Long Short Term Memory*
BPTT	*Back-Propagation Through Time*
BRNN	*Bidirectional Recurrent Neural Network*
CNN	*Convolutional Neural Network*
DBN	*Deep Belief Network*
DRL	*Driven Right Leg Circuit*
ECG	Eletrocardiografia
EEG	Eletroencefalografia
EMG	Eletromiografia
EOG	Eletrooculografia
GRU	*Gated Recurrent Unit*
IAL	*Incremental Attribute Learning*
LSTM	*Long Short Term Memory*
RNN	*Recurrent Neural Network*
RRF	*Rotational Forests*
SAE	*Stacked AutoEncoder*
UCI	Universidade da Califórnia, Irvine

Sumário

1 Introdução

A interface cérebro-computador é um dos campos em ascensão da interação homem-computador, possui um amplo espectro de aplicações, sendo as mais proeminentes as aplicações industriais e médicas. Entre essas aplicações, destacam-se as focadas no mapeamento ou reparação da cognição ou funções motoras de pessoas que possuem seus respectivos sistemas comprometimentos (NAREJO; PASERO; KULSOOM, 2016).

Também em destaque devido ao seu avanço abrupto nas últimas décadas, a área de Aprendizado Profundo se tornou o estado da arte em vários campos, seu conjunto de algoritmos, que tentam aprender hierarquicamente representações não lineares de dados, obteve sucesso em diversas aplicações práticas, inclusive no campo da bioinformática (GOODFELLOW; BENGIO; COURVILLE, 2016; LÄNGKVIST; KARLSSON; LOUTFI, 2014).

Muitos estudos sobre sinais biomédicos, como os obtidos numa eletroencefalografia (EEG), que é o registro da atividade elétrica espontânea do cérebro durante um período de tempo, trouxeram descobertas importantes e úteis para a classificação do estado cognitivo humano (NAREJO; PASERO; KULSOOM, 2016; WANG et al., 2014; SABANCI; KOKLU, 2015). Entre eles estão trabalhos bem sucedidos que utilizaram os sinais do EEG como dado base para classificação do estado dos olhos (abertos ou fechados) afim de detectar crises epiléticas (POLAT; GÜNEŞ, 2007), identificação de característica de estresse (SULAIMAN et al., 2011), detecção de sonolência ao dirigir (YEO et al., 2009), entre outros.

Como os sinais biomédicos representam dados naturalmente sequenciais, diversas técnicas de Aprendizado de Máquina e abordagens estatísticas podem ser empregadas para resolver problemas como os de classificação desses dados (LÄNGKVIST; KARLSSON; LOUTFI, 2014). Entre essas técnicas, destacam-se as Redes Neurais Recorrentes, que possuem uma arquitetura de aprendizado apropriada para analisar dados e que, geralmente, produz resultados promissores.

Este trabalho tem como objetivo propor uma abordagem para a identificação do estado do olho a partir dos sinais cerebrais do EEG usando Redes Neurais Recorrentes e variantes.

Para facilitar a identificação das vantagens e desvantagens da metodologia de aprendizado de máquina utilizada neste trabalho, será utilizado o conjunto de dados de *benchmark EEG Eye State Data Set* (RÖSLER, 2013)[1] que está disponível publicamente.

Este trabalho está está organizado como segue: no Capítulo 2 são apresentados os objetivos do projeto, bem como os objetivos específicos do plano de trabalho em questão; já o Capítulo 3 apresenta conceitos teóricos utilizados, além de uma análise da literatura; no Capítulo 4 são detalhados os procedimentos, recursos e arranjos utilizados no desenvolvimento do trabalho; no Capítulo 5 são expostos e analisados os resultados obtidos a partir dos experimentos desenvolvidos; o Capítulo 6 traz as considerações finais a respeito dos resultados obtidos, além de suas contribuições e limitações; Por fim no Capítulo 7 e Capítulo 8, panoramas de continuidade futura do projeto e as outras atividades realizadas no período de execução do mesmo, são apresentadas, respectivamente.

[1] Disponível em <http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State>.

2
Objetivos

O objetivo geral do projeto é desenvolver modelos de reconhecimento de padrões biométricos, com finalidade de aplicá-los como facilitadores na comunicação de pessoas com comprometimento no sistema motor de produção da fala. Para isso pretende-se:

1. Desenvolver um modelo de máquina de aprendizado sequencial, especializado em reconhecer padrões do eletroencefalograma;
2. Desenvolver um modelo de máquina de aprendizado para visão computacional, especializado em realizar o rastreamento dos olhos.

O escopo deste trabalho limita-se ao desenvolvimento do plano de trabalho descrito no item (1) acima.

2.1 Objetivos Específicos

As etapas necessárias para o cumprimento do plano de trabalho em questão são as seguintes:

- Implementar uma máquina LSTM e variantes;
- Coletar dados de EEG;
- Treinar os modelos;
- Testar e refinar o modelo.

3

Revisão da Literatura

3.1 Eletroencefalografia (EEG)

Eletroencefalografia (EEG) é a medida da atividade elétrica produzida pelo cérebro, registrada por eletrodos colocados no couro cabeludo. Esta atividade é registrada, condicionada e exibida na tela do computador. O EEG é utilizado para diagnosticar lesões cerebrais, tumores e outras anomalias de forma não invasiva, além de ser utilizado em pesquisas de neurociência, ciência cognitiva, psicologia cognitiva, neurolinguística e pesquisa psicofisiológica (NORTHROP, 2012).

As localizações topográficas dos eletrodos são padronizadas e geralmente aderem ao sistema internacional 10-20 (ACHARYA et al., 2016), vide Figura 1. O "10" e o "20" referem-se ao fato de que as distâncias reais entre os eletrodos adjacentes são 10% ou 20% da distância total da frente ou da direita e esquerda do crânio. Cada posição de gravação é identificada por uma combinação de letras e números. Por exemplo, F = Frontal, Fp = Pós frontal, C = Central, P = Parietal, O = Occipital, T = Temporal, A = Lóbulo da orelha, Pg = Faríngea, Z = Zero (linha média). Os números ímpares estão no lado esquerdo da cabeça, e os números pares estão à direita.

Segundo (NORTHROP, 2012), por ser geralmente registrada a partir do couro cabeludo, a atividade elétrica do córtex cerebral deve passar pelas membranas pia e dura-máter, líquido cefalorraquidiano, crânio e couro cabeludo. Por causa dessas estruturas anatômicas, há uma atenuação considerável em relação à atividade elétrica captada pelos eletrodos. Os maiores potenciais de EEG registrados no couro cabeludo são de aproximadamente 150 μV-pico.

Na aferição EEG podem ser usadas configurações bipolares ou unipolares de eletrodos. Na primeira configuração (ver Figura 2, A), a diferença de potencial entre um par de eletrodos é medida. Na segunda (ver Figura 2, B), o potencial de cada eletrodo é comparado a um eletrodo neutro ou à média de todos os eletrodos (média espacial).

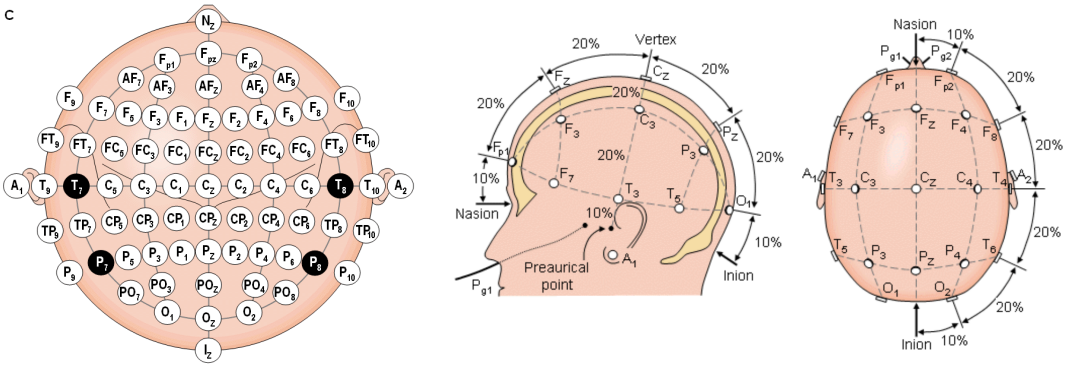

Figura 1 – (Esquerda) Localização e nomenclatura dos eletrodos, de acordo com o padrão da *American Electroencephalographic Society*. (Centro e Direita) Vista lateral esquerda e de topo do posicionamento dos eletrodos de acordo com o padrão. Fonte: (MALMIVUO; PLONSEY, 1995)

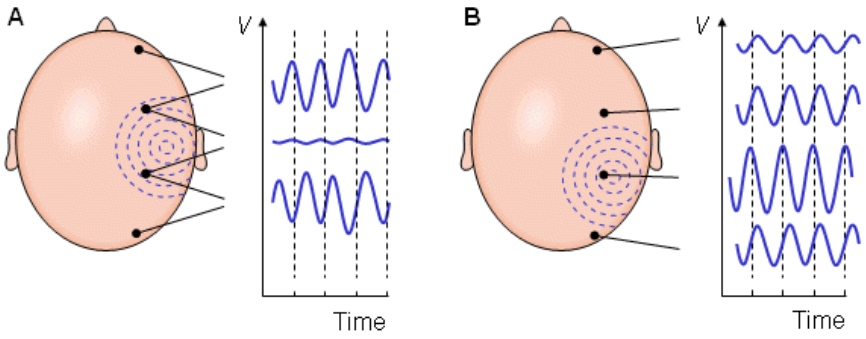

Figura 2 – (A) Configuração bipolar e (B) Configuração unipolar. Fonte: (MALMIVUO; PLONSEY, 1995)

Os ritmos cerebrais estão, tradicionalmente, agrupados em bandas de frequência, que incluem (COHEN, 2014; NORTHROP, 2012):

- As ondas delta (δ) são definidas para frequências de 2 a 4 Hz, ocorrem em adultos em sono profundo;
- As ondas teta (θ) são definidas para frequências de 4 a 8 Hz, ocorrem durante o sono em adultos e em crianças pré-púberes, também pode ser vista durante a meditação;
- As ondas alfa (α) são definidas para frequências de 8 a 14 Hz, ocorrem em adultos conscientes, mas relaxados, com os olhos fechados, desaparece quando os olhos estão abertos e o indivíduo se concentra em alguma tarefa;
- As ondas beta (β) são definidas para frequências de 14 a 30 Hz. As ondas beta são subdivididas nos tipos I e II: as ondas beta tipo I desaparecem durante a atividade mental intensa, enquanto as tipo II aparecem.

Vale ressaltar que não existem limites precisos que definem as bandas de frequência. Além disso, as diferenças individuais nas frequências de pico estão ligadas a uma série de características individuais, incluindo estrutura do cérebro, idade, capacidade de memória de trabalho e composição química do cérebro (COHEN, 2014).

Diferentes padrões de EEG em áreas distintas do cérebro estão intimamente associados com o nível de consciência do indivíduo. À medida que a atividade aumenta, o EEG muda para maior frequência dominante e menor amplitude. Estados de excitação, falta de atenção, sonolência, diferentes estágios do sono, presença de diferentes tipos de drogas e toda uma série de estados clínicos se manifestam distintamente no EEG. Quando os olhos estão fechados, as ondas alfa tendem a dominar o EEG. Quando a pessoa adormece, a frequência de EEG dominante diminui. No sono profundo, o EEG possui amplitudes grandes e frequência baixa (ondas delta) (MALMIVUO; PLONSEY, 1995). Durante a meditação, geralmente ocorrem aumentos de amplitude nas ondas teta e alfa e diminuição da frequência geral (CAHN; POLICH, 2006).

Existem diversas formas de exibir os dados do EEG, entre elas estão a visualização bruta e a espectral (COHEN, 2014). A primeira, chamada de não processada, ou EEG bruto, oferece uma excelente resolução temporal, no entanto, tendências ao longo do tempo são difíceis de avaliar em uma tela de computador, particularmente porque elas se relacionam com mudanças na frequência. A segunda maneira de visualizar o EEG é chamada processada, ou EEG espectral. É chamado processado porque o computador analisa a forma de onda afim de fornecer mais informações, como, por exemplo, mostrar como as frequências do EEG mudam ao longo do tempo.

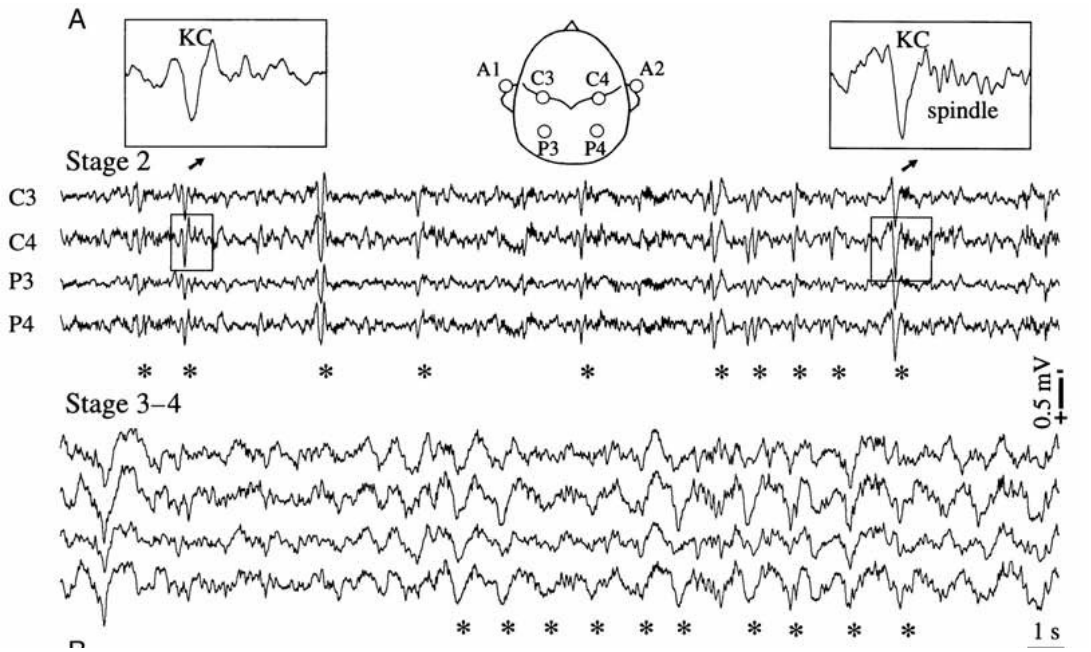

Figura 3 – Exemplo de visualização de EEG (bruto) humano durante o sono natural em diferentes estágios. Fonte: (SCHOMER; SILVA, 2012)

3.2 Circuito de aquisição e condicionamento de sinais biomédicos

O circuito de aquisição e condicionamento foi baseado nos primeiros estágios da plataforma *YouMake* (GOIS, 2017). Esta plataforma foi concebida para a aquisição de diversos sinais fisiológicos existentes de forma didática, versátil, de baixo custo, genérica e de fácil prototipagem.

Os módulos do *YouMake* (ver Figura 4) são os seguintes:

- Entrada dos eletrodos e pré-amplificação – circuito de aquisição de sinais biológicos (entrada dos eletrodos) e amplificação inicial;
- Rejeição do sinal de modo comum – circuito de rejeição do sinal de modo comum (idealmente não existe nenhum ganho nesse bloco);
- Circuito de referência – referência para a medição entre as entradas dos eletrodos, fornece uma redução no modo comum;
- Filtro passa-altas – circuito para remoção do sinal DC;
- Amplificação – circuito para amplificação dos sinais biológicos (varia de acordo com o sinal biológico em questão);
- Filtro passa-baixas – circuito para a filtragem de frequências menores que determinado limiar (varia de acordo com o sinal biológico em questão);
- Filtro rejeita-faixa (opcional) – circuito para a filtragem de frequências de determinado limiar, neste caso 60 Hz (frequência da rede de fornecimento de energia elétrica no Brasil, sua presença é indesejada nos sinais em questão);

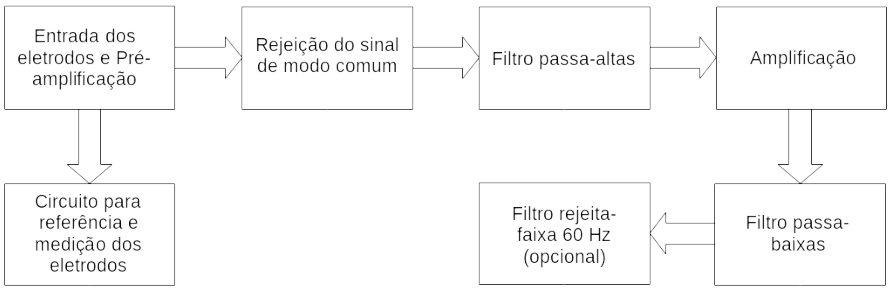

Figura 4 – Diagrama de blocos do *YouMake*. Fonte: (GOIS, 2017)

Os testes de funcionalidade do *YouMake* demonstraram a funcionalidade da plataforma para os sinais do eletrocardiograma (ECG), da eletromiografia (EMG) e do electrooculografia (EOG). As faixas de frequência utilizadas para os testes de funcionalidade variaram entre 0 a 500 Hz, que atendem aos requisitos da aplicação deste trabalho.

3.3 Redes Neurais Recorrentes (RNN)

Uma Rede Neural Recorrente, do inglês *Recurrent Neural Network* (RNN), é uma rede neural artificial em que as conexões entre as unidades ocultas formam ciclos. As Redes Neurais Recorrentes são um tipo específico de rede neural projetada para problemas que envolvem sequências (GRAVES, 2012).

Dada uma rede *perceptron feedforward*, uma rede neural recorrente pode ser pensada como a adição de *loops* à arquitetura. Desse modo, a entrada da RNN inclui também a entrada da etapa anterior. A rede processa a entrada atual, e usa um *loop* de *feedback* para considerar as entradas do passo anterior, também chamado de passado recente, para o contexto. As conexões recorrentes adicionam estado, ou memória, à rede e permitem que ela lide com abstrações mais amplas das sequências de entrada.

Formalmente, uma RNN *vanilla* é definida como segue (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$\mathbf{h}_t = \phi_h(W_x\mathbf{x}_t + W_h\mathbf{h}_{t-1} + b_h) \tag{3.1}$$

$$\mathbf{y}_t = \phi_y(W_y\mathbf{h}_t + b_o) \tag{3.2}$$

Onde $\mathbf{x}_t$, $\mathbf{h}_t$ e $\mathbf{y}_t$ são *arrays* das camadas de entrada, oculta e de saída, respectivamente; W_h, W_x e W_y são matrizes de pesos (conexões entre neurônios); ϕ_h e ϕ_y são as funções de ativação da camada oculta e da camada de saída, respectivamente; e b_h e b_y são o *bias* para a camada oculta e de saída, respectivamente. Os subíndices t referem-se ao período de tempo.

O algoritmo *back-propagation through time* (BPTT) para o treinamento de uma RNN é uma extensão do algoritmo *back-propagation*. Ele foi derivado, observando o *unfolding*, no tempo da rede recorrente numa rede mais simples *feedforward* (GOODFELLOW; BENGIO; COURVILLE, 2016).

O algoritmo BPTT é esquematizado da seguinte forma:

Require: inicialize os pesos da rede (geralmente com valores randômicos pequenos)
  function BACK-PROPAGATION-THROUGH-TIME(a, y)
    desdobrar a rede que contém k instâncias de f
    repeat
      $x =$vetor de magnitude zero;
      for t de 0 até $(n-k)$ **do**
        Defina as entradas da rede como $x, a[t], a[t+1], ..., a[t+k-1]$
        $p =$ propague as entradas para toda a rede em direção à saída (*forward*)
        $e = y[t+k] - p$;
        propague o erro, e, de volta para toda a rede desdobrada (*backward*)

Some as diferenças dos pesos nas k instâncias de f juntas.
Atualize todos os pesos de f e g.
$x = f(x, a[t])$;
end for
until critério de parada satisfeito
return rede com pesos calibrados
end function

Ao lidar com abstrações mais amplas das sequências de entrada, as RNNs *vanilla* podem ficar suscetíveis a dificuldades de treinamento para resolver problemas que exigem o aprendizado de dependências temporais longas. Isso ocorre porque o gradiente da função de perda decai exponencialmente com o tempo (chamado de problema do desaparecimento do gradiente) (GRAVES, 2012). Para tratar esse problema, a literatura relacionada sugere a utilização de um conjunto de "portas"para controlar o fluxo de informações entre as unidades.

3.4 Redes *Long Short Term Memory* (LSTM)

As redes *Long Short Term Memory* (LSTM) são um tipo de RNN com arquitetura apropriada para lidar com sequências e séries temporais, uma rede LSTM pode aprender dependências de longo prazo entre as etapas de tempo destas entradas (HOCHREITER; SCHMIDHUBER, 1997; GRAVES, 2013). Para isso, essas redes mantêm dinamicamente em suas unidades uma célula de memória, além dos portões de entrada, esquecimento e saída que modulam o fluxo de informações dentro da unidade (GREFF et al., 2017).

Estas redes abordam o problema do desaparecimento do gradiente, comumente encontrado nas RNNs, incorporando funções de ativação nas dinâmicas de estado de seus neurônios. Em cada instante de tempo, uma unidade LSTM mantém um vetor oculto h e um vetor de memória c responsável pelo controle de atualizações e saídas de estados (ZAREMBA; SUTSKEVER; VINYALS, 2014).

Formalmente, uma rede LSTM é definida como segue (YAO et al., 2015):

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1}) \tag{3.3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1}) \tag{3.4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1}) \tag{3.5}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t) \tag{3.6}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.7}$$

Onde i_t, f_t e o_t são chamados de portas de entrada, esquecimento e saída, respectivamente; c_t é a memória interna da unidade; $\mathbf{h}_t$ é o estado oculto de saída obtido pela multiplicação da memória com o portão de saída; σ é a função de ativação; W_{xi}, W_{xf}, W_{xc} e W_{xo} são matrizes de pesos da entrada; W_{hi}, W_{hf}, W_{hc} e W_{ho} são matrizes de pesos recorrentes; W_{ci}, W_{cf} e W_{co} são os pesos *peephole*.

3.5 Redes *Gated Recurrent Unit*

As redes *Gated Recurrent Unit* (GRU) podem ser condiseradas uma versão mais simples das redes LSTM (não possuem células de memória dentro de suas unidades), foram propostas por (CHO et al., 2014) para fazer com que cada unidade recorrente capture de forma adaptativa as dependências de diferentes escalas de tempo. Essa arquitetura também utiliza portões que controlam o fluxo de informações dentro do neurônio, no entanto, diferentemente da unidade LSTM, possui apenas dois portões, o de atualização e o de redefinição (CHUNG et al., 2014; YAO et al., 2015).

Formalmente, em relação ao instante de tempo t, uma rede GRU é definida como segue (YAO et al., 2015):

$$h_t = (1 - z_t)h_{t-1} + z_t\hat{h}_t \tag{3.8}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{3.9}$$

$$\hat{h}_t = \tanh\left(W_h x_t + U(r_t \odot h_{t-1})\right) \tag{3.10}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{3.11}$$

Onde z_t e r_t são chamados de portas de atualização e redefinição, respectivamente; $\hat{h}_t$ é a saída candidata; W_z, W_h, W_r, U_z e U_r são matrizes de pesos.

3.6 Redes Neurais Recorrentes Bidirecionais

As arquiteturas mencionadas nas Seções 3.3, 3.4 e 3.5 aprendem representações das sequências a partir de etapas anteriores de tempo. No entanto, dependendo da aplicação, pode ser útil que a rede aprenda com a série temporal completa em cada etapa de tempo, como, por exemplo, em aplicações de reconhecimento de fala e de escrita (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma Rede Neural Recorrente Bidirecional pode ter suas unidades baseadas nas arquiteturas apresentadas anteriormente (Seções 3.3, 3.4 e 3.5), mantendo sempre duas conexões com direções opostas: uma avançando no tempo (lidando com representações anteriores da

sequência), e uma retrocedendo no tempo (lidando com representações futuras da sequência) (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.7 Trabalhos Relacionados

A classificação do estado dos olhos é um tipo de problema de série temporal comum para detectar o estado cognitivo humano. Estudos anteriores apresentam várias abordagens aplicadas na identificação do estado ocular do EEG.

Na literatura, o *corpus* construído em (RÖSLER; SUENDERMANN, 2013), que registrou o estado ocular e o EEG de um indivíduo por 117 segundos, é tido como um problema de referência, e é disponibilizado pelo Repositório de Aprendizagem de Máquina, Universidade da Califórnia, Irvine (UCI) (DHEERU; TANISKIDOU, 2017). No estudo (RÖSLER; SUENDERMANN, 2013), foram empregados 42 métodos diferentes de aprendizados de máquina e abordagens estatísticas utilizando o Weka (HOLMES; DONKIN; WITTEN, 1994) para tentar prever se os olhos de um indivíduo estavam abertos ou fechados. Foi obtida uma taxa de classificação correta de 97,3% com o algoritmo K* (CLEARY; TRIGG, 1995).

Os pesquisadores de (WANG et al., 2014) abordaram o Aprendizado de Atributos Incrementais (IAL), no qual as *features* são gradualmente importadas para o sistema uma por vez para prever a rotulagem de classe. O estudo (HAMILTON; SHAHRYARI; RASHEED, 2015) desenvolveu três diferentes modelos de aprendizagem, o modelo mais preciso foi o construído a partir da combinação *Rotational Forests* (RRF) e K* com uma acurácia de 97.4%. Já o trabalho (KIM; LEE; LIM, 2016) propôs um sistema baseado em um pequeno número de regras Neuro Fuzzy para um problema similar de classificação. O melhor desempenho para o *corpus* em questão foi uma taxa de erro média de 4.0%

No trabalho (NAREJO; PASERO; KULSOOM, 2016), foi proposta a utilização de duas arquiteturas de aprendizado profundo para a tarefa de classificação do estado do olho do EEG. Especificamente *Deep Belief Network* (DBN) e *stacked AutoEncoder* (SAE). O melhor desempenho para o *corpus* em questão foi obtido pelo modelo SAE, com 98.1% de acurácia.

Já o trabalho (BASHIVAN et al., 2015) propôs a utilização de redes híbridas (redes neurais convolucionais e recorrentes) no intuito de absorver as características espacial, espectral e temporal do EEG. Para isso, em vez de representar recursos de EEG bruto como um vetor, transformou os dados em uma sequência de imagens multi espectrais que preservam a topologia e usa esses dados para treinar redes recorrentes e convolucionais profundas para aprender representações robustas a partir da sequência de imagens.

Todos esses trabalhos demonstraram que diversas técnicas de Aprendizado de Máquina e abordagens estatísticas são viáveis na resolução do problema da classificação dos sinais cerebrais que compõem um EEG para a identificação do estado do olho de um indivíduo.

4
Metodologia

A metodologia empregada neste projeto contempla duas fases: a primeira envolve a classificação de padrões no EEG utilizando redes profundas LSTM e suas variantes, e a segunda envolve a preparação de um arranjo experimental para a aquisição e preparação de dados para treinamento de modelos.

4.1 Classificação do EEG

Na primeira etapa serão implementados modelos de redes profundas LSTM em linguagem Python, utilizando a API de redes neurais de alto nível Keras (CHOLLET et al., 2015) com o *backend* do *framework* TensorFlow (ABADI et al., 2015). Os testes iniciais serão realizados com bases de dados disponíveis para download gratuitamente, como a base EEG disponível no repositório UCI [1], intitulada *EEG Eye State Data Set*, na qual todos os dados são de uma medição de EEG contínua com o dispositivo *EEI Neuroheadset Emotiv*[2]. Foram utilizados 14 eletrodos nas posições AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4. Como duração da medição de 117 segundos e o estado do olho foi detectado através de uma câmera durante a medição do EEG e adicionado manualmente ao arquivo depois de analisar os quadros de vídeo. '1' indica o olho fechado e '0' o estado do olho aberto. Todos os valores estão em ordem cronológica com o primeiro valor medido na parte superior dos dados.

Durante os experimentos, a base foi particionada em conjuntos de treinamento e teste com as respectivas divisões de 80% e 20%, com ordem aleatória de seleção. Além disso, também foi utilizada a técnica de validação de modelo *k-fold cross-validation*, com $k = 10$, para avaliar como os resultados serão generalizados para um conjunto de dados independente.

Nos experimentos foram utilizados modelos baseados nas arquiteturas RNN *vanilla*,

[1] Disponível em <http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State>.
[2] Mais informações em <https://www.emotiv.com/epoc/>.

Figura 5 – *Neuroheadset* EMOTIV EPOC+ utilizado para capturar os dados da base. Fonte: (EMOTIV, 2018).

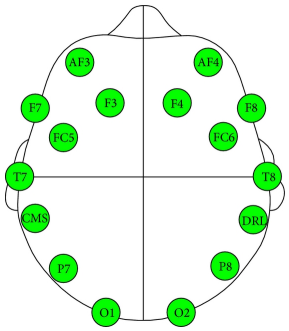

Figura 6 – Posicionamento dos sensores do *neuroheadset*. Fonte: (WANG et al., 2014).

LSTM e GRU, bem como suas variantes bidirecionais. Por fim, os resultados do modelo proposto (apresentando métricas que descrevem o desempenho completo do modelo) são comparados com os de trabalhos relacionados que utilizaram a mesma base de dados.

4.2 Aquisição do EEG e coleta dos dados

Nesta etapa será realizada a montagem da base de dados utilizando o *hardware* desenvolvido, baseado na plataforma *YouMake* (GOIS, 2017). O esquema do circuito e do arranjo experimental pode ser consultado no Apêndice B.

Inicialmente, foram implementados circuitos para dois canais de registro, para os eletrodos Fp1 e Fp2, além dos canais dos eletrodos de referência A1 e A2. Os circuitos eram compostos por componentes discretos plugados numa placa de prototipagem simples. Os eletrodos utilizados são do tipo superficial, descartável, com rebites de prata (Ag/AgCl) e gel condutor de celulose sólido (utilizados no registro de eletrocardiograma).

O objetivo desta etapa é coletar dados suficientes para a aplicação no treinamento dos modelos deste trabalho. O padrão a ser reconhecido será, inicialmente, o mesmo do *corpus* da UCI mencionado na seção anterior.

5

Resultados e discussões

5.1 Análise e tratamento da base de dados *EEG Eye State*

Ao efetuar uma analise exploratória da base de dados notou-se a presença de valores fora do normal, foi realizado o tratamento da base, removendo estes *outliers*[1], a base resultante tratada tem 8172 instâncias válidas de "olho aberto" (‘0’) e 6720 instâncias válidas de "olho fechado" (‘1’).

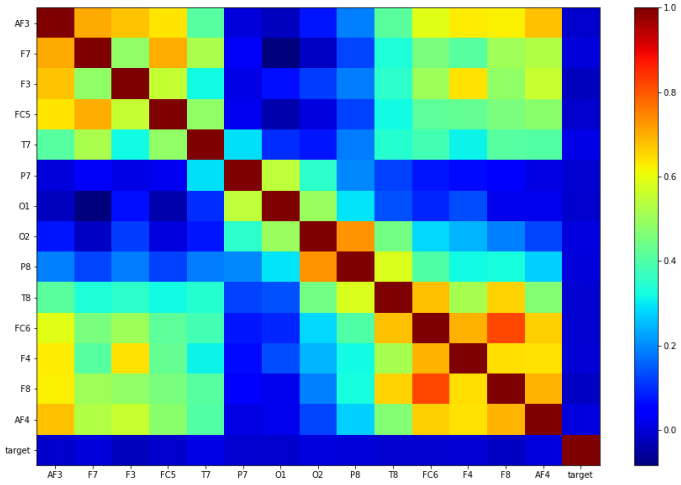

Figura 7 – Matriz de correlação da base de dados representada como um mapa de calor.

Em seguida verificou-se os coeficientes de correlação (matriz de correlação), ver Figura 7, para tentar identificar as *features* (colunas do conjunto de dados) que têm uma alta correlação

[1] *Outliers* são valores extremos muito além das outras observações do conjunto de dados.

com a variável de *target*[2]. No caso desta base de dados, as correlações se mantém próximas a zero, ou seja, provavelmente não se encontrará uma base ótima para representar os dados de forma compacta, isso pode ser observado verificando as cores frias na última linha (e na última coluna, já que se trata de uma matriz simétrica), tendo em vista que a classe ou variável de decisão (*target*) está associada à última posição nesta matriz.

Também quantificou-se a relação de dependência linear (matriz de dispersão) entre as *features*, que fornece um resumo gráfico das relações no conjunto de dados, verificou-se que, os histogramas das variáveis parecem estar normalmente distribuídas (vide Apêndice C). Além disso, as relações entre alguns pares sugerem um padrão linear (pontos mais quentes no gráfico de calor da matriz de correlação).

Antes de utilizar os dados nos modelos, em uma etapa complementar à remoção de *outliers* do conjunto de dados, foi realizado um escalonamento das instâncias para que permanecessem entre um valor mínimo e máximo, neste caso entre zero e um. Isto porquê, em geral, os algoritmos de aprendizado se beneficiam da padronização do conjunto de dados.

O código relacionado a esta etapa está disponível no Apêndice A.4.

5.2 Modelos propostos para a base de dados *EEG Eye State*

Todos os modelos utilizados, possuíam três camadas ocultas, com 28 unidades cada, com *dropout* de 50% entre as camadas, é importante destacar também a utilização da regularização L2 ($\lambda = 0.0001$) nas matrizes recorrentes das unidades. A utilização do *dropout* e da regularização foram medidas preventivas contra *overfitting* dos modelos.

A rede RNN *vanilla* obteve acurácia de $81.53\,\% \pm 3.20\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 1. Já sua variante bidirecional obteve acurácia de $88.59\,\% \pm 1.47\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 2.

Tabela 1 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo RNN.

	precision	**recall**	**f1-score**	**support**
eye opened	0,897	0,876	0,886	1635
eye closed	0,854	0,877	0,865	1344
avg/total	**0,877**	**0,877**	**0,877**	**2979**

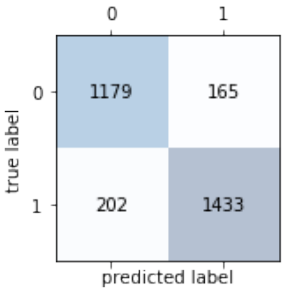

Figura 8 – Matriz de confusão do modelo RNN.

[2] Variável de *target*, no contexto deste trabalho, é a variável que é ou deve ser a saída. Por exemplo, pode ser 0 ou 1 binário (tarefa de classificação) ou pode ser uma variável contínua (tarefa de regressão).

Tabela 2 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo BRNN.

	precision	**recall**	**f1-score**	**support**
eye opened	0,898	0,943	0,920	1635
eye closed	0,926	0,870	0,897	1344
avg/total	**0,911**	**0,910**	**0,910**	**2979**

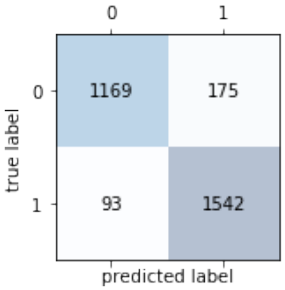

Figura 9 – Matriz de confusão do modelo BRNN.

Os códigos relacionados aos modelos baseados na RNN e BRNN *vanilla* estão disponíveis no Apêndice A.8 e A.11, respectivamente.

A rede LSTM obteve acurácia de $94.35\,\% \pm 0.91\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 3. Já sua variante bidirecional obteve acurácia de $96.13\,\% \pm 0.54\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 4.

Tabela 3 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo LSTM.

	precision	**recall**	**f1-score**	**support**
eye opened	0,935	0,956	0,946	1635
eye closed	0,945	0,920	0,932	1344
avg/total	**0,940**	**0,940**	**0,939**	**2979**

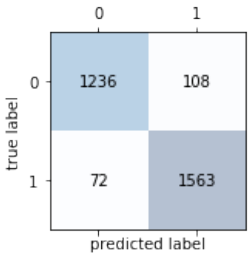

Figura 10 – Matriz de confusão do modelo LSTM.

Tabela 4 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo BLSTM.

	precision	**recall**	**f1-score**	**support**
eye opened	0,967	0,955	0,961	1635
eye closed	0,946	0,960	0,953	1344
avg/total	**0,957**	**0,957**	**0,957**	**2979**

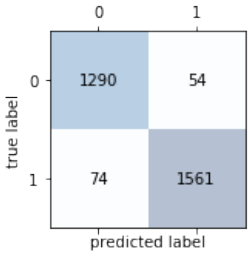

Figura 11 – Matriz de confusão do modelo BLSTM.

Os códigos relacionados aos modelos baseados na LSTM e BLSTM estão disponíveis no Apêndice A.7 e A.10, respectivamente.

Por fim, a rede GRU obteve acurácia de $93.73\,\% \pm 1.00\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 5. Já sua variante bidirecional obteve acurácia de $96.13\,\% \pm 0.76\,\%$ (*10-fold*), as demais métricas são apresentadas na Tabela 6.

Tabela 5 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo GRU.

	precision	**recall**	**f1-score**	**support**
eye opened	0,931	0,956	0,944	1635
eye closed	0,945	0,914	0,929	1344
avg/total	**0,937**	**0,937**	**0,937**	**2979**

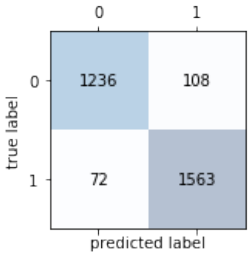

Figura 12 – Matriz de confusão do modelo GRU.

Tabela 6 – Resumo da precisão, *recall*, *F1-score* para cada classe do modelo BGRU.

	precision	**recall**	**f1-score**	**support**
eye opened	0,961	0,960	0,961	1635
eye closed	0,952	0,952	0,952	1344
avg/total	**0,957**	**0,957**	**0,957**	**2979**

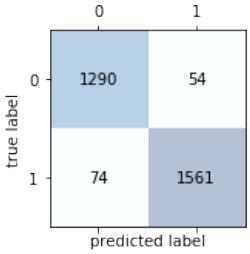

Figura 13 – Matriz de confusão do modelo BGRU.

Os códigos relacionados aos modelos baseados na GRU e BGRU estão disponíveis no Apêndice A.6 e A.9, respectivamente.

Baseado nestes resultados, constata-se que os modelos de redes recorrentes bidirecionais com unidades LSTM e GRU obtiveram o melhor desempenho na tarefa de classificação do *corpus*. As variantes unidirecionais destas arquiteturas apresentaram desempenho inferior, permanecendo próximos aos índices das redes bidirecionais. Já o modelo baseado em células RNN *vanilla* obteve o pior desempenho, atribui-se a isto sua arquitetura mais simples e limitada, como discutido anteriormente na Seção 3.3. Sua variante bidirecional, a BRNN, também apresentou desempenho superior se comparada a RNN.

O estado da arte para classificação deste conjunto de dados obteve resultados superiores aos obtidos neste trabalho ao custo de maior esforço computacional e consequentemente maior complexidade, a exemplo dos classificadores K* (RÖSLER; SUENDERMANN, 2013) e K*+RRF (HAMILTON; SHAHRYARI; RASHEED, 2015) obtêm resultados superiores a 90%. Também são destacados os classificadores *Neuro-Fuzzy* (KIM; LEE; LIM, 2016), que apresentaram resultados similares aos de (RÖSLER; SUENDERMANN, 2013; HAMILTON; SHAHRYARI; RASHEED, 2015). Por fim, o modelo proposto por (NAREJO; PASERO; KULSOOM, 2016), utilizando *Stacked AutoEncoders* (SAE), apresenta o melhor desempenho. Os modelos propostos neste trabalho possuem arquitetura simplificada e visam atender minimamente ao compromisso de melhor desempenho e melhor classificação possível.

Além das técnicas utilizadas na literatura para alcançar tais resultados, outros trabalhos relacionados (com diferentes conjuntos de dados) também se destacam, como os que utilizam

Tabela 7 – Comparação com os modelos da literatura.

Modelo	**Sucesso na classificação (%)**
MLP (SABANCI; KOKLU, 2015)	56.4
RNN	81.5
kNN (SABANCI; KOKLU, 2015)	84.1
BRNN	88.6
GRU	93.7
LSTM	94.3
BGRU	96.1
BLSTM	96.1
Neuro-Fuzzy (KIM; LEE; LIM, 2016)	96.0
K* (RÖSLER; SUENDERMANN, 2013)	97.3
K*+RRF (HAMILTON; SHAHRYARI; RASHEED, 2015)	97.4
SAE (NAREJO; PASERO; KULSOOM, 2016)	98.9

redes híbridas CNN/LSTM para absorver as características espacial, espectral e temporal do EEG (BASHIVAN et al., 2015).

5.3 Implementação e utilização do equipamento de aquisição

Para os experimentos iniciais de captura de dados de EEG foram implementados os primeiros módulos do *YouMake* para verificar sua funcionalidade, adaptar conexões, calibrar o circuito e para familiarização com o *hardware* envolvido.

Para um registro simplificado foi sugerido o uso de 3 eletrodos na parte frontal da cabeça (Fp1, Fpz e Fp2) e uma referência, que normalmente são ligados nos lóbulos das duas orelhas, em A1 e A2 (referência bi-auricular), ou em CZ (referência unipolar). Por conveniência foi utilizado a referência bi-auricular.

No experimento, foram utilizados 3 eletrodos inicialmente, dois para referência (em A1 e A2) e o terceiro para o sinal (em Fp1). Os resultados (ver Figura 15) são a tensão de saída do amplificador operacional AD620AN como o circuito de referência (porção inferior da parte direita da Figura 17), obtidos de um indivíduo acordado e concentrado em uma atividade.

A Figura 16 exibe o diagrama de blocos do circuito completo, incluindo a captura e o condicionamento do sinal. Na Figura 17 estão representados os circuitos que implementam os blocos deste diagrama. Os valores dos componentes utilizados seguiram, inicialmente, os definidos por (GOIS, 2017) para a captura do EMG de superfície, já que engloba todas as faixas de frequência do EEG. Seguindo as abordagens de (GOIS, 2017), foram simulados e posteriormente implementados:

- um circuito utilizando amplificadores operacionais TL084CN, que tem custo baixo por unidade e deixam o circuito maior; e

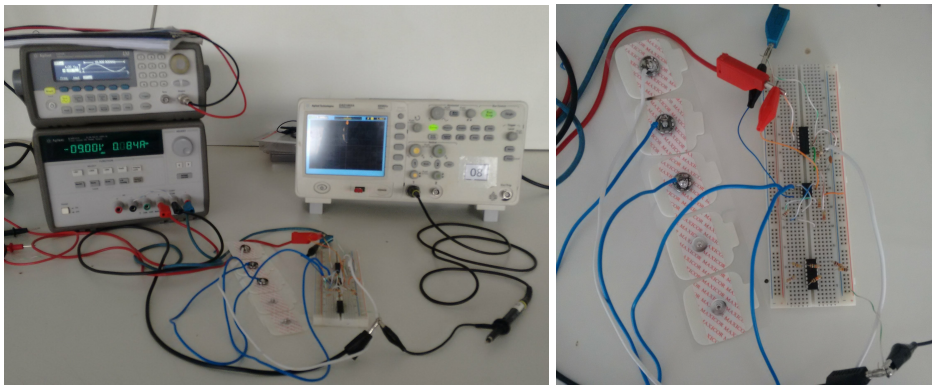

Figura 14 – (Esquerda) Arranjo experimental do primeiro experimento. (Direita) Implementação da parte inferior do circuito exibido na Figura 17 (Direita).

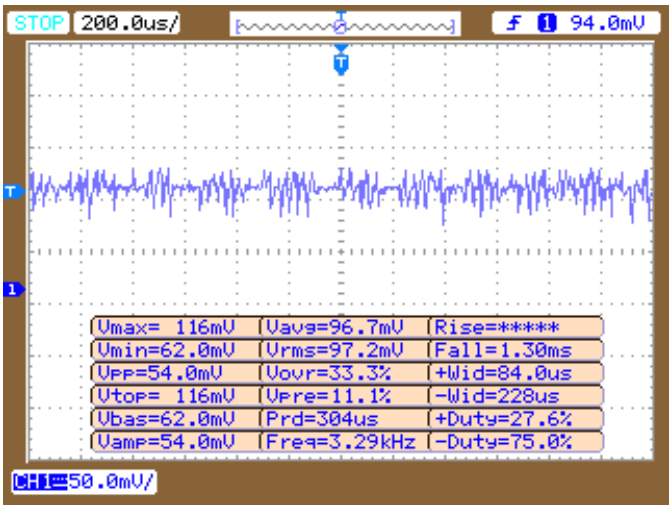

Figura 15 – Sinal obtido pelo circuito implementado no experimento 1.

- um circuito utilizando amplificadores de instrumentação AD620AN (que tem custo elevado por unidade) e amplificadores operacionais TL084CN, que deixam o circuito mais compacto.

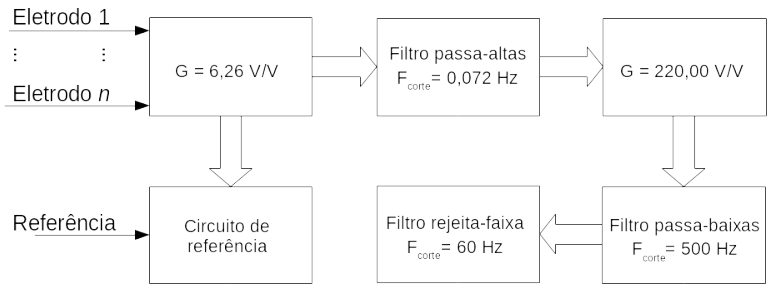

Figura 16 – Diagrama de blocos das etapas de aquisição e condicionamento do sinal do EEG. Fonte: Adaptado de (GOIS, 2017).

A etapa posterior a aquisição e condicionamento do sinal do EEG foi a digitalização do

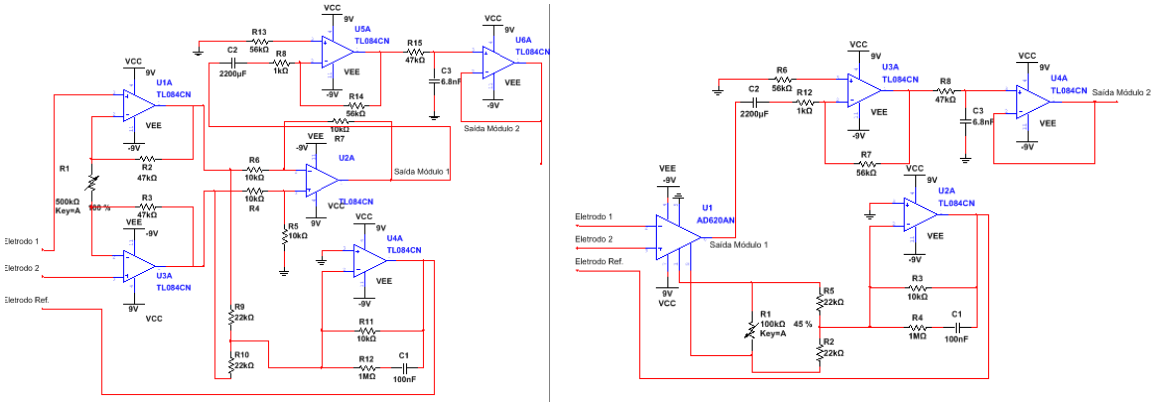

Figura 17 – (Esquerda) Simulação do circuito utilizando amplificadores operacionais TL084CN. (Direita) Simulação do circuito utilizando amplificadores de instrumentação AD620AN e amplificadores operacionais TL084CN. O circuito com o AD620 oferece um melhor desempenho em relação ao projeto com amplificadores operacionais discretos, além de um tamanho menor, menos componentes e uma corrente de alimentação inferior (cerca de 10 vezes menor) (ANALOG DEVICES, 2011).

mesmo. Para isso foi utilizado um Arduino Nano[3], uma variante de baixo custo da plataforma de prototipagem eletrônica de *hardware* livre (também seguindo os mesmos princípios da plataforma *YouMake*), projetada com um microcontrolador que oferece suporte a entrada/saída embutido e uma linguagem de programação padrão (extensão de C/C++) (D'AUSILIO, 2012). O esquema completo do sistema de aquisição, registro e visualização está disponível no Apêndice B.

Nas iterações seguintes dos experimentos foram realizados testes de funcionalidade do sistema para assegurar seu funcionamento dentro das especificações. Foram substituídos os cabos dos eletrodos por versões blindadas (em quase toda sua extensão) e os conectores dos eletrodos ganharam uma proteção plástica (para evitar ruídos devido ao contato com os mesmos), ver Figura 18.

Apesar disso, nos experimentos subsequentes obtidos dados não satisfatórios, a interferência no sinal era predominante, apresentando maior amplitude em relação amplitude do biopotencial do EEG. Este fenômeno é bem documentado na literatura e se deve, entre outras razões, ao o corpo do paciente, a disposição dos componentes discretos e cabos na *protoboard* e a porção sem blindagem dos cabos dos eletrodos também pode atuar como uma antena que capta interferências eletromagnéticas, especialmente ruído de 60 Hz de linhas de energia elétrica. Essa interferência pode mascarar os sinais biológicos, em especial o sinal do EEG (com a maior faixa de interesse entre 2 e 30 Hz), dificultando a captura isolada deste.

Afim de minimizar esse impacto no sinal desejado foram testadas algumas combinações de filtros rejeita faixa (60 Hz), depois dos filtros passa-alta e baixa (faixa de interesse), antes e depois dos filtros passa-alta e baixa (faixa de interesse) e também um filtro passa-faixa de interesse (frequência final em 50 Hz); mas a atenuação da frequência de interferência não teve

[3] Mais informações disponíveis em <https://www.arduino.cc/en/Guide/ArduinoNano>

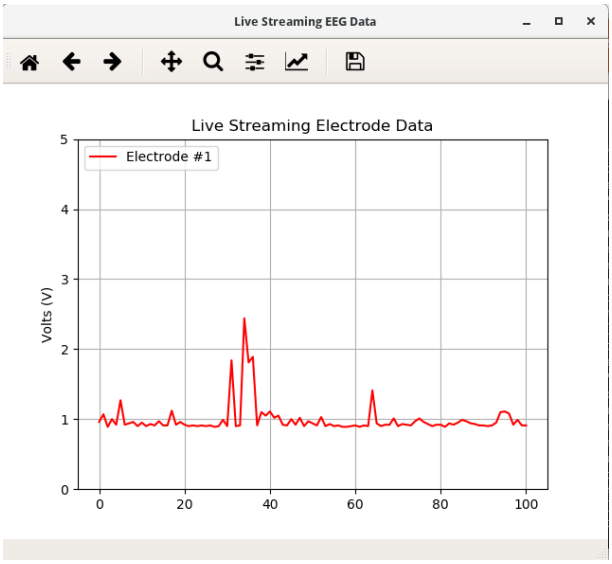

Figura 19 – Janela da aplicação de visualização do sinal de um eletrodo de EEG.

sua amplitude reduzida significativamente em relação a do biopotencial amplificado do EEG (permaneceram na mesma ordem de grandeza, cerca de 200 mV). E por outro lado, quanto maior a atenuação da frequência de interferência, mais significativo era o impacto na faixa de frequência de interesse. Também foi utilizada a alimentação por baterias, mas ao conectar o Arduino a porta USB do computador para a transmissão dos dados a interferência voltava a atuar no sinal adquirido.

Um solução passível de implementação é apresentada no esquema de *hardware* do projeto *OpenEEG*[4] (GRIFFITH, 2006), no qual é utilizado um conversor DC/DC para a alimentação do circuito de captura e acopladores ópticos de alta velocidade nas vias de comunicação serial do microcontrolador, o que garante o isolamento entre o circuito de captura e o computador. Além disso, contribuem para a obtenção de melhores resultados e minimização do problema: a disposição do circuito numa placa de circuito impresso, a utilização de *laptops* desconectados da rede elétrica, ou *smartphones*, a utilização de cabos totalmente blindados de qualidade e de eletrodos específicos.

Tendo em vista o caráter obrigatório da solução deste problema, pode não ser possível atender totalmente ao intuito principal da plataforma *YouMake* de ser versátil e de baixo custo, mas certamente será menos custosa do que uma solução comercial, como os apresentados por (GRIFFITH, 2006; MAHAJAN; MORSHED; BIDELMAN, 2016).

[4] Mais informações em <http://openeeg.sourceforge.net/doc/modeeg/modeeg.html>.

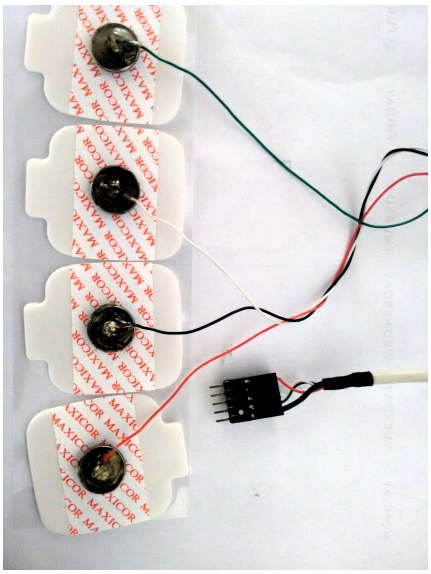

Figura 18 – Eletrodos superficiais descartáveis com rebites de prata (Ag/AgCl) e gel condutor de celulose sólido (para ECG), conectores e cabos de dados utilizados.

6

Considerações finais

Neste projeto, uma abordagem de classificação de séries temporais baseada em redes neurais recorrentes é proposta para a identificação do estado ocular do paciente utilizando seus dados de EEG. A abordagem utilizada focou na utilização de modelos baseados em redes LSTM e suas variantes, além da RNN *vanilla*, para efetuar tal classificação.

Foram efetuadas etapas de tratamento dos dados brutos comumente mencionadas na literatura, afim de transformar o conjuntos de dados para que o conteúdo deste seja melhor aproveitado no treinamento dos modelos.

Os resultados obtidos pelos modelos são próximos ou equivalentes aos apresentados pela literatura, indicando que os modelos de redes recorrentes bidirecionais com unidades LSTM e GRU são aplicáveis na classificação de séries temporais de EEG.

Também neste projeto, foram dedicados esforços para a utilização de *hardware* baseado na plataforma *YouMake* para captura de dados de EEG de forma simples e de baixo custo. No entanto a implementação deste equipamento atendendo aos requisitos de custo e simplicidade não apresentou resultados satisfatórios para sinais de baixa amplitude, como o do EEG. As capturas registradas com um ou mais canais apresentavam comportamentos destoantes dos observados na literatura para sinais normais de EEG, carregados de artefatos e com alto nível de interferência externa.

7

Perspectivas

Os resultados deste projeto são os passos iniciais para a aplicação do reconhecimento de padrões biomédicos utilizando aprendizado profundo em dispositivos baseados na plataforma *YouMake*. A execução do plano de trabalho em questão demonstrou a aplicação dos modelos num *corpus* específico, com apenas duas categorias; e realizou experimentos para a coleta de apenas dois canais de EEG utilizando diversos protótipos de dispositivos de aquisição.

Existem muitas melhorias a serem aplicadas tanto nos modelos de aprendizado de máquina, quanto no protótipo do dispositivo de aquisição (principalmente). Para os modelos, sugere-se a investigação de novas abordagens, como as arquiteturas híbridas mencionadas anteriormente. Também é sugerida a experimentação de técnicas para melhorar as métricas dos modelos apresentados, como a de *data augmentation*, já que a quantidade de dados do *corpus* utilizado é limitada e esses algoritmos tendem a se beneficiar com o aumento do conjunto de dados, além disso, pode ser explorada a extração de características do sinal como novas *features*, por exemplo.

Em relação a implementação da plataforma, podem ser aplicadas sugestões da literatura para minimização da interferência eletromagnética no sinal do EEG, para proteção do circuito de aquisição contra picos de tensão e para substituição do eletrodos e cabos por tipos específicos blindados, ainda tentando guardar o compromisso com o custo final (de acordo com as intenções da plataforma *YouMake*). Além disso, podem ser exploradas iniciativas da comunidade *open source*, como a OpenEEG[1], que visa disponibilizar gratuitamente esquemas de *hardware* e *software* para dispositivos EEG do tipo "faça você mesmo".

[1] Mais informações em <http://openeeg.sourceforge.net/doc/modeeg/modeeg.html>

8

Outras Atividades

Participação como ouvinte no Minicurso de Introdução à Aprendizagem Profunda com o *Tensorflow*, promovido pelo Laboratório para Universalização do Desenvolvimento, Inovação e Inteligência Computacional (LUDIICO)[1], do Departamento de Computação da Universidade Federal de Sergipe. O curso que ocorreu nos dias 22, 24, 29 e 31 de agosto de 2017, teve carga horária total de 12 horas e abordou tópicos como: redes neurais artificiais, *perceptron* multicamadas, redes neurais convolucionais e redes neurais recorrentes.

[1] Mais informações em: <http://dgp.cnpq.br/dgp/espelhogrupo/6184846048078907>

Referências

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>. Citado na página 21.

ACHARYA, J. N. et al. American clinical neurophysiology society guideline 2: guidelines for standard electrode position nomenclature. *The Neurodiagnostic Journal*, Taylor & Francis, v. 56, n. 4, p. 245–252, 2016. Citado na página 13.

ANALOG DEVICES. *Low Cost Low Power Instrumentation Amplifier*. [S.l.], 2011. Rev. H. Citado 2 vezes nas páginas 3 e 29.

BASHIVAN, P. et al. Learning representations from eeg with deep recurrent-convolutional neural networks. *arXiv preprint arXiv:1511.06448*, 2015. Citado 2 vezes nas páginas 20 e 27.

CAHN, B. R.; POLICH, J. Meditation states and traits: Eeg, erp, and neuroimaging studies. *Psychological bulletin*, American Psychological Association, v. 132, n. 2, p. 180, 2006. Citado na página 15.

CHO, K. et al. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. Citado na página 19.

CHOLLET, F. et al. *Keras*. 2015. <https://keras.io>. Citado na página 21.

CHUNG, J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. Citado na página 19.

CLEARY, J. G.; TRIGG, L. E. K*: An instance-based learner using an entropic distance measure. In: *12th International Conference on Machine Learning*. [S.l.: s.n.], 1995. p. 108–114. Citado na página 20.

COHEN, M. X. *Analyzing neural time series data: theory and practice*. [S.l.]: MIT Press, 2014. Citado 2 vezes nas páginas 14 e 15.

D'AUSILIO, A. Arduino: A low-cost multipurpose lab equipment. *Behavior Research Methods*, v. 44, n. 2, p. 305–313, Jun 2012. ISSN 1554-3528. Disponível em: <https://doi.org/10.3758/s13428-011-0163-z>. Citado na página 29.

DHEERU, D.; TANISKIDOU, E. K. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 20.

EMOTIV. *EMOTIV EPOC+ 14 Channel Mobile EEG Headset*. 2018. Disponível em: <https://www.emotiv.com/product/emotiv-epoc-14-channel-mobile-eeg/#tab-description>. Citado 2 vezes nas páginas 3 e 22.

GOIS, D. A. S. *Youmake: Uma plataforma didática, de baixo custo, genérica e de fácil prototipagem para aquisição e condicionamento de sinais biológicos*. Dissertação (Mestrado) — Universidade Federal de Sergipe, São Cristóvão, SE, Brasil, 2017. Citado 5 vezes nas páginas 3, 16, 22, 27 e 28.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Citado 4 vezes nas páginas 10, 17, 19 e 20.

GRAVES, A. *Supervised Sequence Labelling with Recurrent Neural Networks*. 1. ed. [S.l.]: Springer-Verlag Berlin Heidelberg, 2012. (Studies in Computational Intelligence 385). ISBN 3642247962,9783642247965. Citado 2 vezes nas páginas 17 e 18.

GRAVES, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. Citado na página 18.

GREFF, K. et al. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, IEEE, 2017. Citado na página 18.

GRIFFITH, A. An exploration of the openeeg project. *CHG Wright's BioData Systems*, Citeseer, 2006. Citado na página 30.

HAMILTON, C. R.; SHAHRYARI, S.; RASHEED, K. M. Eye state prediction from eeg data using boosted rotational forests. In: IEEE. *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. [S.l.], 2015. p. 429–432. Citado 3 vezes nas páginas 20, 26 e 27.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 18.

HOLMES, G.; DONKIN, A.; WITTEN, I. H. Weka: A machine learning workbench. In: IEEE. *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. [S.l.], 1994. p. 357–361. Citado na página 20.

KIM, Y.; LEE, C.; LIM, C. Computing intelligence approach for an eye state classification with eeg signal in bci. In: WORLD SCIENTIFIC. *Software Engineering and Information Technology: Proceedings of the 2015 International Conference on Software Engineering and Information Technology (SEIT2015)*. [S.l.], 2016. p. 265–270. Citado 3 vezes nas páginas 20, 26 e 27.

LÄNGKVIST, M.; KARLSSON, L.; LOUTFI, A. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, Elsevier, v. 42, p. 11–24, 2014. Citado na página 10.

MAHAJAN, R.; MORSHED, B. I.; BIDELMAN, G. M. Design and validation of a wearable "drl-less"eeg using a novel fully-reconfigurable architecture. In: *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.: s.n.], 2016. p. 4999–5002. ISSN 1557-170X. Citado na página 30.

MALMIVUO, J.; PLONSEY, R. *Bioelectromagnetism: principles and applications of bioelectric and biomagnetic fields*. [S.l.]: Oxford University Press, USA, 1995. Citado 3 vezes nas páginas 3, 14 e 15.

NAREJO, S.; PASERO, E.; KULSOOM, F. Eeg based eye state classification using deep belief network and stacked autoencoder. *International Journal of Electrical and Computer Engineering (IJECE)*, v. 6, n. 6, p. 3131–3141, 2016. Citado 4 vezes nas páginas 10, 20, 26 e 27.

NORTHROP, R. B. *Analysis and Application of Analog Electronic Circuits to Biomedical Instrumentation, Second Edition*. 2nd ed. ed. [S.l.]: CRC Press, 2012. (Biomedical engineering series (Boca Raton Fla.)). ISBN 978-1-4398-6743-3,1439867437. Citado 2 vezes nas páginas 13 e 14.

POLAT, K.; GÜNEŞ, S. Classification of epileptiform eeg using a hybrid system based on decision tree classifier and fast fourier transform. *Applied Mathematics and Computation*, Elsevier, v. 187, n. 2, p. 1017–1026, 2007. Citado na página 10.

RÖSLER, O. *EEG Eye State Data Set*. 2013. Disponível em: <https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State>. Citado na página 11.

RÖSLER, O.; SUENDERMANN, D. A first step towards eye state prediction using eeg. *Proc. of the AIHLS*, 2013. Citado 3 vezes nas páginas 20, 26 e 27.

SABANCI, K.; KOKLU, M. The classification of eye state by using knn and mlp classification models according to the eeg signals. *International Journal of Intelligent Systems and Applications in Engineering*, v. 3, n. 4, p. 127–130, 2015. Citado 2 vezes nas páginas 10 e 27.

SCHOMER, D. L.; SILVA, F. L. D. *Niedermeyer's electroencephalography: basic principles, clinical applications, and related fields*. [S.l.]: Lippincott Williams & Wilkins, 2012. Citado 2 vezes nas páginas 3 e 15.

SULAIMAN, N. et al. Novel methods for stress features identification using eeg signals. *International Journal of Simulation: Systems, Science and Technology*, United Kingdom Simulation Society, v. 12, n. 1, p. 27–33, 2011. Citado na página 10.

WANG, T. et al. Eeg eye state identification using incremental attribute learning with time-series classification. *Mathematical Problems in Engineering*, Hindawi, v. 2014, 2014. Citado 4 vezes nas páginas 3, 10, 20 e 22.

YAO, K. et al. Depth-gated recurrent neural networks. arxiv preprint. *arXiv preprint arXiv:1508.03790*, v. 9, 2015. Citado 2 vezes nas páginas 18 e 19.

YEO, M. V. et al. Can svm be used for automatic eeg detection of drowsiness during car driving? *Safety Science*, Elsevier, v. 47, n. 1, p. 115–124, 2009. Citado na página 10.

ZAREMBA, W.; SUTSKEVER, I.; VINYALS, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. Citado na página 18.

Apêndices

APÊNDICE A – Códigos fonte

Listing A.1 – Leitura da tensão de um eletrodo e envio dos dados pela porta serial (Arduino Nano)

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a
    ↪ voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1024.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

Listing A.2 – Apresentação dos dados da porta serial.

```
import serial
import numpy
import time
import csv
import matplotlib.pyplot as plt
from drawnow import *

eletrodo1 = []
arduinoData = serial.Serial('/dev/ttyUSB0', 19200)
plt.ion() # matplotlib interactive mode to plot live data
plt.gcf().canvas.set_window_title('Live Streaming EEG Data')
counter = 0

def makeFig():
    plt.ylim(0,5)
```

```
    plt.title('Live Streaming Electrode Data')
    plt.grid(True)
    plt.ylabel('Volts (V)')
    plt.plot(eletrodo1, 'r', label='Electrode #1')
    plt.legend(loc='upper left')

    # add more electrodes here


while True:
    while (arduinoData.inWaiting() == 0): # Wait here until
       ↪ there is data
        pass

    arduinoString = arduinoData.readline()
    eletrodo1_voltage =float( arduinoString)
    eletrodo1.append(eletrodo1_voltage)

    with open("electrode_data.csv","a") as f:
        writer = csv.writer(f,delimiter=",")
        writer.writerow([time.time(), eletrodo1_voltage])

    drawnow(makeFig)
    plt.pause(.000001)

    counter = counter + 1
    if(counter > 100):
        eletrodo1.pop(0)
```

Listing A.3 – *Download* do *dataset*.

```
import os.path
import sys

import pandas as pd
import requests
from scipy.io import arff


def load():
```

```
    # check if data file exists locally
    if not os.path.isfile('EEG Eye State.arff'):
        url = 'https://archive.ics.uci.edu/ml/machine-learning-
            ↪ databases/00264/EEG%20Eye%20State.arff'
        try:
            r = requests.get(url, allow_redirects=True)
            open('EEG Eye State.arff', 'wb').write(r.content)
        except IOError as e:
            print("I/O error({0}): {1}".format(e.errno, e.
                ↪ strerror))
        except:
            print("Unexpected error:", sys.exc_info()[0])
            raise

    # read data from local arff file
    data = arff.loadarff('EEG Eye State.arff')

    df = pd.DataFrame(data[0])

    # insert columns names
    df.columns = ['AF3', 'F7', 'F3', 'FC5', 'T7', 'P7', 'O1',
                  'O2', 'P8', 'T8', 'FC6', 'F4', 'F8', 'AF4',
                  'eyeDetection']
    return df
```

Listing A.4 – Tratamendo dos dados do *dataset*.

```
import numpy as np
from sklearn import preprocessing


def transform(df):
    # reject NaN entries
    df.dropna()

    # remove outliers
    df1 = df[df.loc[:, df.columns != 'eyeDetection']
        .apply(lambda x: np.abs(x - x.mean()) / x.std() < 3).
            ↪ all(axis=1)]
```

```
    # standardize data
    min_max_scaler = preprocessing.MinMaxScaler()
    X = min_max_scaler.fit_transform(df1.iloc[0:, 0:13].values)

    # define target
    Y = df1.iloc[0:, 14].values.astype(int)

    return X, Y
```

Listing A.5 – Apresentação das métricas do modelo.

```
import matplotlib.pyplot as plt
from IPython.display import SVG
from keras.utils import plot_model
from keras.utils.vis_utils import model_to_dot
from sklearn.metrics import classification_report, \
    average_precision_score, precision_recall_curve, \
    precision_score, recall_score, f1_score, \
    confusion_matrix, roc_auc_score, accuracy_score, \
    roc_curve, auc


def plot_overal_progress(history):
    # Plot overall progress
    plt.title('Accuracy over epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(history.history['acc'], label='train')
    plt.plot(history.history['val_acc'], label='val')
    plt.grid(True, which='both', axis='both')

    # plt.minorticks_on()
    plt.tight_layout()

    plt.legend()
    plt.show()

    plt.title('Loss over epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
```

```
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='val')
    plt.grid(True, which='both', axis='both')
    plt.tight_layout()
    plt.legend()
    plt.show()


def print_metrics_prec_rec_f1(model, X_test, y_test):
    y_pred = model.predict_classes(X_test)
    print('Precision: %.3f' % precision_score(y_true=y_test,
       ↪ y_pred=y_pred))
    print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=
       ↪ y_pred))
    print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))


def show_confusion_matrix(model, X_test, y_test, savefig=False)
   ↪ :
    y_pred = model.predict_classes(X_test)
    confmat = confusion_matrix(y_true=y_test, y_pred=y_pred,
       ↪ labels=[1, 0])
    # print(confmat)
    fig, ax = plt.subplots(figsize=(2.5, 2.5))
    ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(confmat.shape[0]):
        for j in range(confmat.shape[1]):
            ax.text(x=j, y=i, s=confmat[i, j], va='center', ha=
               ↪ 'center')

    plt.xlabel('predicted label')
    plt.ylabel('true label')

    plt.tight_layout()
    if savefig:
        plt.savefig('./figures/confusion_matrix.png', dpi=300)
    plt.show()
```

```
def print_metrics_roc_acc(model, X_test, y_test):
    y_pred = model.predict_classes(X_test)
    print('ROC AUC: %.3f' % roc_auc_score(y_true=y_test,
       ↪ y_score=y_pred))
    print('Accuracy: %.3f' % accuracy_score(y_true=y_test,
       ↪ y_pred=y_pred))


def show_roc_curve(model, X_test, y_test):
    y_pred = model.predict_classes(X_test)
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes = 1
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i
           ↪ ])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(),
       ↪ y_pred.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure()
    lw = 2

    plt.plot(fpr[0], tpr[0], color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc
                ↪ [0])

    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--
       ↪ ')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    # plt.title('Receiver operating characteristic example')
```

```
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()


def show_model(model):
    # plot_model(model, to_file='model.png')
    plot_model(model, to_file='model_plot.png', show_shapes=
        ↪ False, show_layer_names=False)
    SVG(model_to_dot(model, show_shapes=True, show_layer_names=
        ↪ True).create(prog='dot', format='svg'))


def print_classif_report(model, X_test, y_test):
    y_pred = model.predict_classes(X_test)
    target_names = ['eye opened', 'eye closed']
    print(classification_report(y_test, y_pred, target_names=
        ↪ target_names, digits=3))


def show_prec_recal_curve(model, X_test, y_test):
    y_pred = model.predict_classes(X_test)
    average_precision = average_precision_score(y_test, y_pred)

    precision, recall, _ = precision_recall_curve(y_test,
        ↪ y_pred)

    plt.step(recall, precision, color='b', alpha=0.2,
             where='post')
    plt.fill_between(recall, precision, step='post', alpha=0.2,
                     color='b')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.
        ↪ format(
         average_precision))
```

Listing A.6 – Treinamento da rede GRU

```
import random

import dataload
import dataprep
import modelinfo
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, GRU
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)

unique, counts = np.unique(y_short, return_counts=True)
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
      y_short, y_short.shape)

# ### Model
# define model parameters
samples = X_short.shape[0]  # how many trials of eeg data
n_features = X_short.shape[1]  # how many channels of eeg in
    ↪ each sample
time_steps = 1  # how many ms was each sample run for

# use strat. shuffle split to get indices for test and training
    ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
```

```
    ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

# take the indices generated by stratified shuffle split and
    ↪ make
# the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

# reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
    ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
    ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(GRU(28, return_sequences=True,
              recurrent_regularizer=l1_l2(0.0001, 0.0001),
              input_shape=(time_steps, n_features)))
model.add(GRU(28, input_shape=(time_steps, n_features),
              recurrent_regularizer=l1_l2(0.0001, 0.0001),
              return_sequences=True))
model.add(Dropout(0.5))
model.add(GRU(28, input_shape=(time_steps, n_features),
              recurrent_regularizer=l1_l2(0.0001, 0.0001)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
    ↪ patience=50,
                           verbose=0, mode='auto')
```

```
terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
    ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.7 – Treinamento da rede

```
import random

import dataload
import dataprep
import modelinfo
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, LSTM
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)

unique, counts = np.unique(y_short, return_counts=True)
```

```
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
      y_short, y_short.shape)

### Model
define model parameters
samples = X_short.shape[0]  how many trials of eeg data
n_features = X_short.shape[1]  how many channels of eeg in
   ↪ each sample
time_steps = 1  how many ms was each sample run for

use strat. shuffle split to get indices for test and training
   ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
   ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

take the indices generated by stratified shuffle split and
   ↪ make
the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
   ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
   ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(LSTM(28, return_sequences=True,
               recurrent_regularizer=l1_l2(0.0001, 0.0001),
               input_shape=(time_steps, n_features)))
```

```
model.add(LSTM(28, input_shape=(time_steps, n_features),
               recurrent_regularizer=l1_l2(0.0001, 0.0001),
               return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(28, input_shape=(time_steps, n_features),
               recurrent_regularizer=l1_l2(0.0001, 0.0001)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
   ↪ patience=50,
                           verbose=0, mode='auto')
terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
   ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.8 – Treinamento da rede

```
import random

import dataload
import dataprep
import modelinfo
import numpy as np
```

```
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, SimpleRNN
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)

unique, counts = np.unique(y_short, return_counts=True)
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
      y_short, y_short.shape)

# ### Model
# define model parameters
samples = X_short.shape[0]  # how many trials of eeg data
n_features = X_short.shape[1]  # how many channels of eeg in
   ↪ each sample
time_steps = 1  # how many ms was each sample run for

# use strat. shuffle split to get indices for test and training
   ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
   ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

# take the indices generated by stratified shuffle split and
   ↪ make
# the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
```

```
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

# reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
   ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
   ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(SimpleRNN(28, return_sequences=True,
                    recurrent_regularizer=l1_l2(0.0001, 0.0001)
                       ↪ ,
                    input_shape=(time_steps, n_features)))
model.add(SimpleRNN(28, input_shape=(time_steps, n_features),
                    recurrent_regularizer=l1_l2(0.0001, 0.0001)
                       ↪ ,
                    return_sequences=True))
model.add(Dropout(0.5))
model.add(SimpleRNN(28, input_shape=(time_steps, n_features),
                    recurrent_regularizer=l1_l2(0.0001, 0.0001)
                       ↪ ))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
   ↪ patience=50,
                           verbose=0, mode='auto')
terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
```

```
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
    ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.9 – Treinamento da rede BGRU

```
import random

import dataload
import dataprep
import modelinfo
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, GRU, Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)

unique, counts = np.unique(y_short, return_counts=True)
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
```

```
      y_short, y_short.shape)

# ### Model
# define model parameters
samples = X_short.shape[0]  # how many trials of eeg data
n_features = X_short.shape[1]  # how many channels of eeg in
   ↪ each sample
time_steps = 1  # how many ms was each sample run for

# use strat. shuffle split to get indices for test and training
   ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
   ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

# take the indices generated by stratified shuffle split and
   ↪ make
# the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

# reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
   ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
   ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(Bidirectional(GRU(28, return_sequences=True,
                            recurrent_regularizer=l1_l2(0.0001,
                               ↪  0.0001),
                            input_shape=(time_steps, n_features
                               ↪ ))))
model.add(Bidirectional(GRU(28, input_shape=(time_steps,
   ↪ n_features),
```

```
                                recurrent_regularizer=l1_l2(0.0001,
                                    ↪  0.0001),
                                return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(GRU(28, input_shape=(time_steps,
    ↪ n_features),
                                recurrent_regularizer=l1_l2(0.0001,
                                    ↪  0.0001))))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
    ↪ patience=50, verbose=0, mode='auto')

terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
    ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.10 – Treinamento da rede BLSTM

```
import random

import dataload
import dataprep
```

```
import modelinfo
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, LSTM, Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)

unique, counts = np.unique(y_short, return_counts=True)
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
      y_short, y_short.shape)

# ### Model
# define model parameters
samples = X_short.shape[0]  # how many trials of eeg data
n_features = X_short.shape[1]  # how many channels of eeg in
   ↪ each sample
time_steps = 1  # how many ms was each sample run for

# use strat. shuffle split to get indices for test and training
   ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
   ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

# take the indices generated by stratified shuffle split and
   ↪ make
```

```
# the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

# reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
   ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
   ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(Bidirectional(LSTM(28, return_sequences=True,
                             recurrent_regularizer=l1_l2
                                ↪ (0.0001, 0.0001),
                             input_shape=(time_steps,
                                ↪ n_features))))
model.add(Bidirectional(LSTM(28, input_shape=(time_steps,
   ↪ n_features),
                             recurrent_regularizer=l1_l2
                                ↪ (0.0001, 0.0001),
                             return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(28, input_shape=(time_steps,
   ↪ n_features),
                             recurrent_regularizer=l1_l2
                                ↪ (0.0001, 0.0001))))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
   ↪ patience=50, verbose=0, mode='auto')
```

```
terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
   ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.11 – Treinamento da rede BRNN

```
import random

import dataload
import dataprep
import modelinfo
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout, SimpleRNN,
   ↪ Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedShuffleSplit

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# ### Load the Data
df = dataload.load()
# Clean dataset
X_short, y_short = dataprep.transform(df)
```

```
unique, counts = np.unique(y_short, return_counts=True)
print(dict(zip(unique, counts)))

print(df.shape)
print(X_short, X_short.shape, '\n\n',
      y_short, y_short.shape)

### Model
define model parameters
samples = X_short.shape[0]  how many trials of eeg data
n_features = X_short.shape[1]  how many channels of eeg in
   ↪ each sample
time_steps = 1  how many ms was each sample run for

use strat. shuffle split to get indices for test and training
   ↪  data
sss = StratifiedShuffleSplit(n_splits=2, test_size=0.2,
   ↪ random_state=seed)
sss.get_n_splits(X_short, y_short)

take the indices generated by stratified shuffle split and
   ↪ make
the test and training datasets
for train_index, test_index in sss.split(X_short, y_short):
    X_train, X_test = X_short[train_index], X_short[test_index]
    y_train, y_test = y_short[train_index], y_short[test_index]

reshape data to properly fit in tensors
X_train = X_train.reshape(X_train.shape[0], time_steps, X_train
   ↪ .shape[1])
y_train = y_train.reshape(y_train.shape[0], time_steps)

X_test = X_test.reshape(X_test.shape[0], time_steps, X_test.
   ↪ shape[1])
y_test = y_test.reshape(y_test.shape[0], time_steps)

model = Sequential()
model.add(Bidirectional(SimpleRNN(28, return_sequences=True,
```

```
                                    recurrent_regularizer=l1_l2
                                        ↪ (0.0001, 0.0001),
                                    input_shape=(time_steps,
                                        ↪ n_features))))
model.add(Bidirectional(SimpleRNN(28, input_shape=(time_steps,
    ↪ n_features),
                                    recurrent_regularizer=l1_l2
                                        ↪ (0.0001, 0.0001),
                                    return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(SimpleRNN(28, input_shape=(time_steps,
    ↪ n_features),
                                    recurrent_regularizer=l1_l2
                                        ↪ (0.0001, 0.0001))))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
    ↪ patience=50, verbose=0, mode='auto')

terminate_NaN = TerminateOnNaN()

history = model.fit(X_train, y_train, validation_split=0.25,
                    batch_size=28, epochs=1000, verbose=1,
                    callbacks=[early_stop, terminate_NaN])

score = model.evaluate(X_test, y_test, batch_size=28)
print("Accuracy: %.2f%%" % (score[1] * 100))

model.save('eeg_eye_state_model_rnn.h5')  # creates a HDF5 file
    ↪  'my_model.h5'
# model = load_model('eeg_eye_state_model_rnn.h5')
model.summary()
modelinfo.plot_overal_progress(history)
modelinfo.print_classif_report(model, X_test, y_test)
```

Listing A.12 – Validação cruzada do modelo GRU

```
# coding: utf-8

import random

import dataload
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import GRU
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# Load dataset
df = dataload.load()
# Clean dataset
X, Y = dataprep.transform(df)

# ### Model
# full dataset parameters

# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
```

```
model.add(GRU(28, return_sequences=True,
              recurrent_regularizer=l1_l2(0.0001, 0.0001),
              input_shape=(time_steps, n_features)))
model.add(GRU(28, input_shape=(time_steps, n_features),
              recurrent_regularizer=l1_l2(0.0001, 0.0001),
              return_sequences=True))
model.add(Dropout(0.5))
model.add(GRU(28, input_shape=(time_steps, n_features),
              recurrent_regularizer=l1_l2(0.0001, 0.0001)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='loss', min_delta=0,
   ↪ patience=20, verbose=1, mode='auto')

terminate_NaN = TerminateOnNaN()

model.fit(X[train].reshape(X[train].shape[0], time_steps, X
   ↪ [train].shape[1]),
          Y[train].reshape(Y[train].shape[0], time_steps),
          batch_size=28,
          epochs=1000,
          verbose=0,
          callbacks=[early_stop, terminate_NaN])

# evaluate the model
scores = model.evaluate(X[test].reshape(X[test].shape[0],
   ↪ time_steps, X[test].shape[1]),
                        Y[test].reshape(Y[test].shape[0],
                           ↪ time_steps),
                        batch_size=28,
                        verbose=1)

print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
   ↪ 100))
```

```
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
    ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
    ↪ file 'my_model.h5'
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

Listing A.13 – Validação cruzada do modelo LSTM

```
# coding: utf-8

import random

import dataload
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import LSTM
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# Load dataset
df = dataload.load()
# Clean dataset
X, Y = dataprep.transform(df)
```

```
# ### Model
# full dataset parameters

# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(LSTM(28, return_sequences=True,
                   recurrent_regularizer=l1_l2(0.0001, 0.0001),
                   input_shape=(time_steps, n_features)))
    model.add(LSTM(28, input_shape=(time_steps, n_features),
                   recurrent_regularizer=l1_l2(0.0001, 0.0001),
                   return_sequences=True))
    model.add(Dropout(0.5))
    model.add(LSTM(28, input_shape=(time_steps, n_features),
                   recurrent_regularizer=l1_l2(0.0001, 0.0001))
                      ↪ )
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer='nadam',
                  metrics=['accuracy'])

    early_stop = EarlyStopping(monitor='loss', min_delta=0,
       ↪ patience=20, verbose=1, mode='auto')

    terminate_NaN = TerminateOnNaN()

    model.fit(X[train].reshape(X[train].shape[0], time_steps, X
```

```
        ↪ [train].shape[1]),
                Y[train].reshape(Y[train].shape[0], time_steps),
                batch_size=28,
                epochs=1000,
                verbose=0,
                callbacks=[early_stop, terminate_NaN])

    # evaluate the model
    scores = model.evaluate(X[test].reshape(X[test].shape[0],
        ↪ time_steps, X[test].shape[1]),
                            Y[test].reshape(Y[test].shape[0],
                                ↪ time_steps),
                            batch_size=28,
                            verbose=1)

    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
        ↪ 100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
    ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
    ↪ file 'my_model.h5'
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

Listing A.14 – Validação cruzada do modelo RNN

```
# coding: utf-8

import random

import dataload
```

```
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import SimpleRNN
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# Load dataset
df = dataload.load()
# Clean dataset
X, Y = dataprep.transform(df)

# ### Model
# full dataset parameters

# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(SimpleRNN(28, return_sequences=True,
                        recurrent_regularizer=l1_l2(0.0001,
                          ↪ 0.0001),
                        input_shape=(time_steps, n_features)))
    model.add(SimpleRNN(28, input_shape=(time_steps, n_features
       ↪ ),
```

```
                    recurrent_regularizer=l1_l2(0.0001,
                        ↪ 0.0001),
                    return_sequences=True))
model.add(Dropout(0.5))
model.add(SimpleRNN(28, input_shape=(time_steps, n_features
    ↪ ),
                    recurrent_regularizer=l1_l2(0.0001,
                        ↪ 0.0001)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='loss', min_delta=0,
    ↪ patience=20, verbose=1, mode='auto')

terminate_NaN = TerminateOnNaN()

model.fit(X[train].reshape(X[train].shape[0], time_steps, X
    ↪ [train].shape[1]),
          Y[train].reshape(Y[train].shape[0], time_steps),
          batch_size=28,
          epochs=1000,
          verbose=0,
          callbacks=[early_stop, terminate_NaN])

# evaluate the model
scores = model.evaluate(X[test].reshape(X[test].shape[0],
    ↪ time_steps, X[test].shape[1]),
                        Y[test].reshape(Y[test].shape[0],
                            ↪ time_steps),
                        batch_size=28,
                        verbose=1)

print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
    ↪ 100))
cvscores.append(scores[1] * 100)
```

```
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
    ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
    ↪ file 'my_model.h5'
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

Listing A.15 – Validação cruzada do modelo BGRU

```
# coding: utf-8

import random

import dataload
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import GRU, Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# Load dataset
df = dataload.load()
print(df)
# Clean dataset
X, Y = dataprep.transform(df)
```

```
# ### Model
# full dataset parameters

# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Bidirectional(GRU(28, return_sequences=True,
                                recurrent_regularizer=l1_l2
                                   ↪ (0.0001, 0.0001),
                                input_shape=(time_steps,
                                   ↪ n_features))))
    model.add(Bidirectional(GRU(28, input_shape=(time_steps,
       ↪ n_features),
                                recurrent_regularizer=l1_l2
                                   ↪ (0.0001, 0.0001),
                                return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(Bidirectional(GRU(28, input_shape=(time_steps,
       ↪ n_features),
                                recurrent_regularizer=l1_l2
                                   ↪ (0.0001, 0.0001))))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer='nadam',
                  metrics=['accuracy'])

    early_stop = EarlyStopping(monitor='loss', min_delta=0,
```

```
        ↪ patience=20,
                                    verbose=1, mode='auto')

    terminate_NaN = TerminateOnNaN()

    model.fit(X[train].reshape(X[train].shape[0], time_steps, X
        ↪ [train].shape[1]),
              Y[train].reshape(Y[train].shape[0], time_steps),
              batch_size=28,
              epochs=1000,
              verbose=1,
              callbacks=[early_stop, terminate_NaN])

    # evaluate the model
    scores = model.evaluate(X[test].reshape(X[test].shape[0],
        ↪ time_steps, X[test].shape[1]),
                            Y[test].reshape(Y[test].shape[0],
                                ↪ time_steps),
                            batch_size=28,
                            verbose=1)

    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
        ↪ 100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
    ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
    ↪ file 'my_model.h5'
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

Listing A.16 – Validação cruzada do modelo BLSTM

```
# coding: utf-8

import random

import dataload
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import LSTM, Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)

# Load dataset
df = dataload.load()
# Clean dataset
X, Y = dataprep.transform(df)

# ### Model
# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Bidirectional(LSTM(28, return_sequences=True,
                                 recurrent_regularizer=l1_l2
```

```
                                    ↪ (0.0001, 0.0001),
                                input_shape=(time_steps,
                                    ↪ n_features))))
model.add(Bidirectional(LSTM(28, input_shape=(time_steps,
    ↪ n_features),
                                recurrent_regularizer=l1_l2
                                    ↪ (0.0001, 0.0001),
                                return_sequences=True)))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(28, input_shape=(time_steps,
    ↪ n_features),
                                recurrent_regularizer=l1_l2
                                    ↪ (0.0001, 0.0001))))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='loss', min_delta=0,
    ↪ patience=20,
                           verbose=1, mode='auto')

terminate_NaN = TerminateOnNaN()

model.fit(X[train].reshape(X[train].shape[0], time_steps, X
    ↪ [train].shape[1]),
          Y[train].reshape(Y[train].shape[0], time_steps),
          batch_size=28,
          epochs=1000,
          verbose=0,
          callbacks=[early_stop, terminate_NaN])

# evaluate the model
scores = model.evaluate(X[test].reshape(X[test].shape[0],
    ↪ time_steps, X[test].shape[1]),
                        Y[test].reshape(Y[test].shape[0],
                            ↪ time_steps),
```

```
                                batch_size=28,
                                verbose=1)

    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
       ↪ 100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
   ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
   ↪ file 'my_model.h5'
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
   ↪  show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

Listing A.17 – Validação cruzada do modelo BRNN

```
# coding: utf-8

import random

import dataload
import dataprep
import numpy as np
from keras.callbacks import EarlyStopping, TerminateOnNaN
from keras.layers import Dense, Dropout
from keras.layers import SimpleRNN, Bidirectional
from keras.models import Sequential
from keras.regularizers import l1_l2
from sklearn.model_selection import StratifiedKFold

# setting the random seed for reproducibility
seed = 42
random.seed(seed)
```

```
# Load dataset
df = dataload.load()
# Clean dataset
X, Y = dataprep.transform(df)

# ### Model
# full dataset parameters

# define model parameters
samples = 14980  # how many trials of eeg data
n_features = 13  # how many channels of eeg in each sample
time_steps = 1  # how many ms was each sample run for

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
cvscores = []

for train, test in kfold.split(X, Y):
    model = Sequential()
    model.add(Bidirectional(SimpleRNN(28, return_sequences=True
       ↪ ,
                                      recurrent_regularizer=
                                         ↪ l1_l2(0.0001,
                                         ↪ 0.0001),
                                      input_shape=(time_steps,
                                         ↪ n_features))))
    model.add(Bidirectional(SimpleRNN(28, input_shape=(
       ↪ time_steps, n_features),
                                      recurrent_regularizer=
                                         ↪ l1_l2(0.0001,
                                         ↪ 0.0001),
                                      return_sequences=True)))
    model.add(Dropout(0.5))
    model.add(Bidirectional(SimpleRNN(28, input_shape=(
       ↪ time_steps, n_features),
                                      recurrent_regularizer=
                                         ↪ l1_l2(0.0001,
```

```
                                                    ↪ 0.0001))))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer='nadam',
                  metrics=['accuracy'])

    early_stop = EarlyStopping(monitor='loss', min_delta=0,
       ↪ patience=20,
                               verbose=1, mode='auto')

    terminate_NaN = TerminateOnNaN()

    model.fit(X[train].reshape(X[train].shape[0], time_steps, X
       ↪ [train].shape[1]),
              Y[train].reshape(Y[train].shape[0], time_steps),
              batch_size=28,
              epochs=1000,
              verbose=0,
              callbacks=[early_stop, terminate_NaN])

    # evaluate the model
    scores = model.evaluate(X[test].reshape(X[test].shape[0],
       ↪ time_steps, X[test].shape[1]),
                            Y[test].reshape(Y[test].shape[0],
                               ↪ time_steps),
                            batch_size=28,
                            verbose=1)

    print("%s: %.2f%%" % (model.metrics_names[1], scores[1] *
       ↪ 100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(
   ↪ cvscores)))

# model.save('eeg_eye_state_model_kfold.h5')  # creates a HDF5
   ↪ file 'my_model.h5'
```

```
# model = load_model('eeg_eye_state_model_kfold.h5')

# plot_model(model, to_file='model.png')
# plot_model(model, to_file='model_plot.png', show_shapes=True,
    ↪ show_layer_names=True)
# SVG(model_to_dot(model).create(prog='dot', format='svg'))

# model.summary()
```

APÊNDICE B – Esquema geral do circuito de aquisição implementado

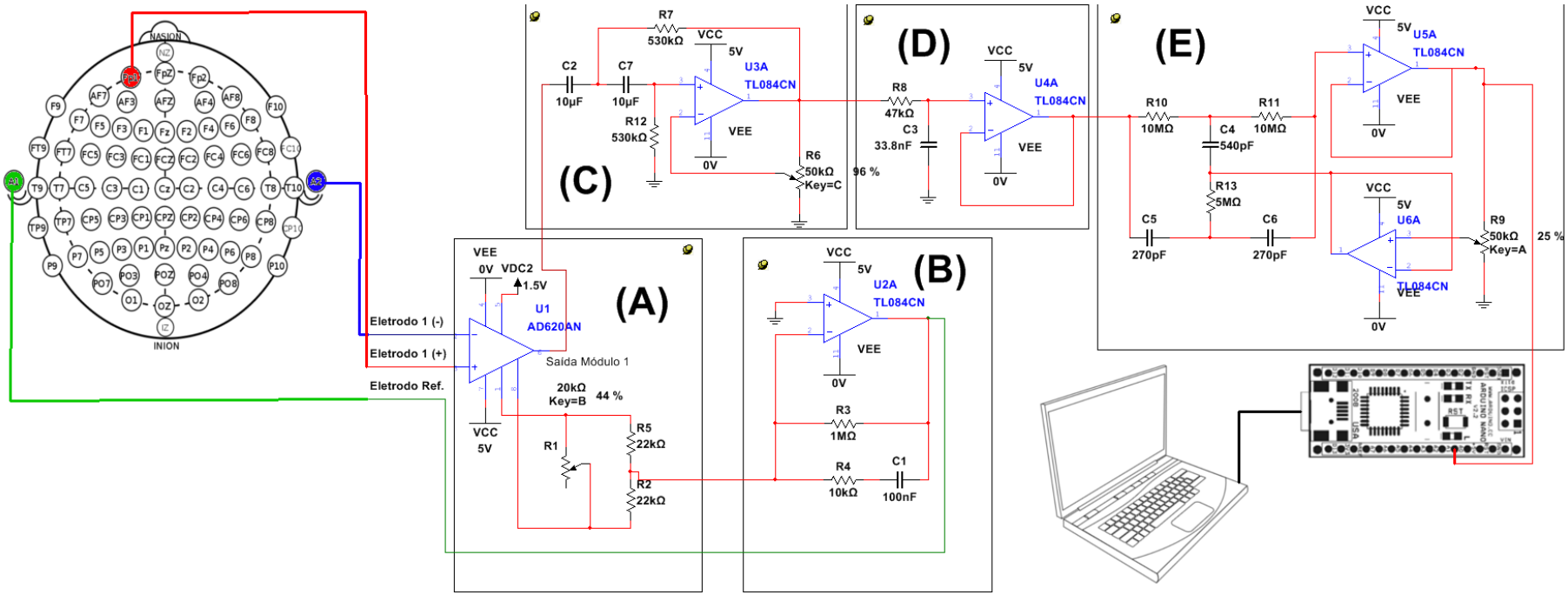

Figura 20 – Esquema geral do sistema de aquisição. Posicionamento dos eletrodos. **(A)** Amplificador operacional AD620. **(B)** *Driven Right Leg Circuit* (DRL). **(C)** Filtro passa altas $f_c = 0.03$ Hz. **(D)** filtro passa baixas $f_c = 100$ Hz. **(E)**. Filtro rejeita faixa $f_c = 60$ Hz. Arduino Nano configurado para enviar a leitura da porta analógica para a porta serial conectado (USB) a um computador para registro e visualização dos dados.

APÊNDICE C – Matriz de dispersão da base de dados

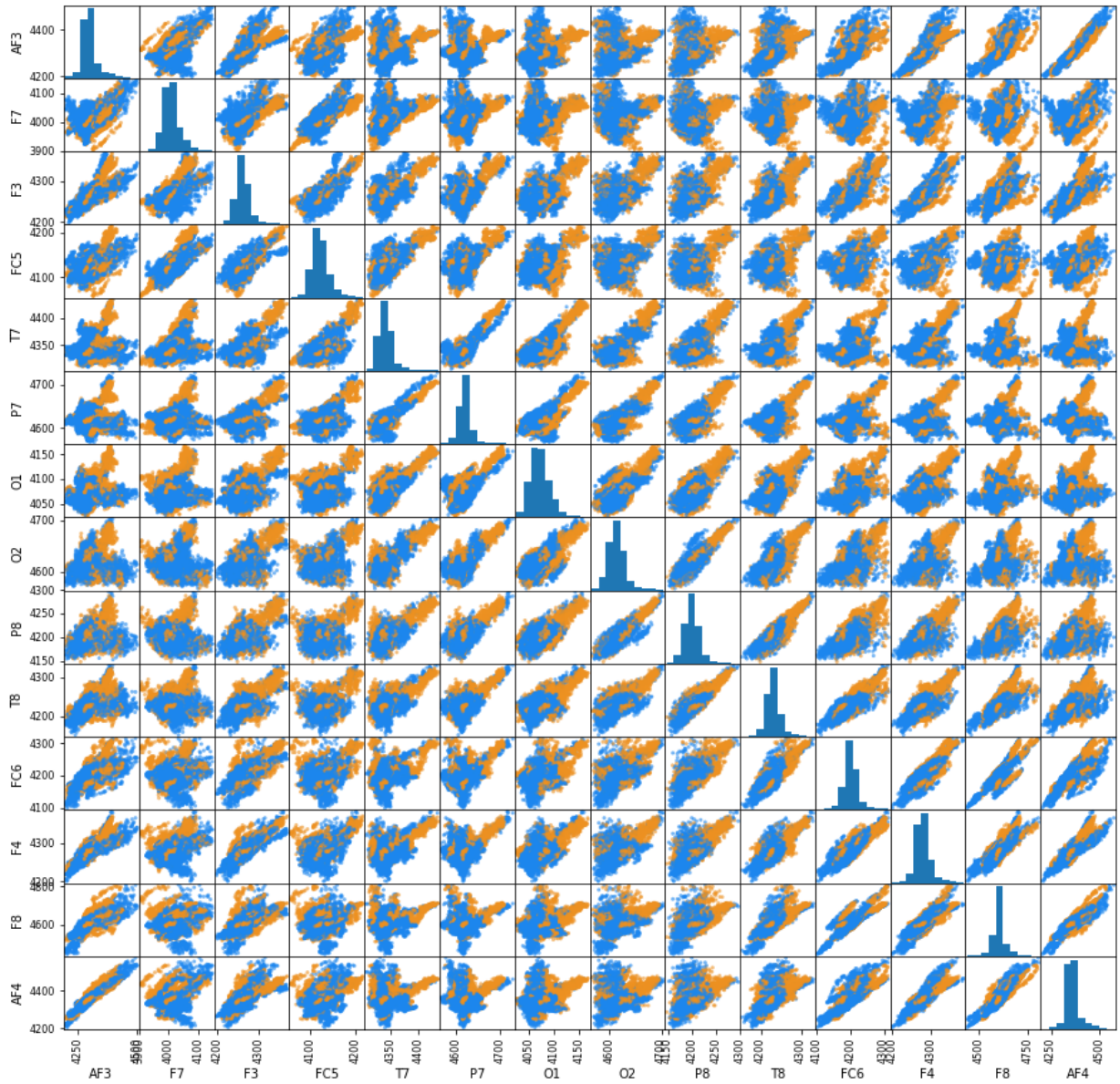

Figura 21 – Matriz de dispersão da base de dados *EEG Eye State Data Set*.